A Scheduling Tool for Conditionally Independent Temporal Preferences

A Scheduling Tool for Conditionally Independent Temporal Preferences

Roberto Micalizio, Gianluca Torta
Dipartimento di informatica
Università di Torino
Torino, Italy
Email: {roberto.micalizio,gianluca.torta}@unito.it

***Abstract*—It is well known that the problem of finding a feasible schedule for a partially ordered set of tasks can be formulated as a Disjunctive Temporal Problem (DTP). In case we want to find a schedule by taking preferences into account, there exist extensions to DTPs that augment them by associating numeric costs to the violation of individual temporal constraints; however, such extensions make the restrictive assumption that the costs associated with constraints are independent of one another. In this paper we propose a further extension, which enables the designer to specify (directional) *dependencies* between the preferences associated with the constraints. Such preferences are represented by exploiting Utility Difference Networks (UDNs), that define objective functions whose structure reflects conditional independencies among the nodes of the network. The paper describes the branch-and-bound algorithm at the core of the scheduling tool we have implemented for solving this new class of problems. We also present and discuss encouraging experimental results collected in two different test domains.**

***Keywords*-scheduling; preferences; DTP**

I. Introduction

Since the seminal work by Dechter et al. [1], Temporal Constraint Satisfaction Problems (TCSPs) have drawn the attention of several AI researchers, and many problem formulations have been proposed ever since. In particular, the notion of Disjunctive Temporal Problems (DTPs) [2] has been introduced to overcome the limits of Simple Temporal Problems (STPs) [1] by enabling the specification of temporal constraints consisting of disjuncts, each of which represents a temporal interval within which legal solutions can be found. This class of problems is expressive enough to model scheduling problems [3], as well as other problems of interest in AI (e.g., diagnosis [4], [5]). In [6], [7] the notion of *flexible schedule* is introduced. Differently from a usual scheduling, in a flexible schedule tasks are not associated with precise starting and ending times, but with intervals within which they can take place.

Research on DTPs has recently been focused on how to address temporal *preferences* (i.e., *soft constraints*). Intuitively, a soft constraint allows one to express preferences on the distance between any two time points. For instance, in a calendar management scenario [8], relevant time points are reasonably the start and end times of the activities to be scheduled. Soft constraints can therefore be used to express the preference that some activities should last as long as possible, or that the distance between the ending of an activity and the starting of the subsequent one should be minimized. While solving an STP or a DTP usually comes down to verifying the satisfiability of the (hard) constraints specified in the problem, solving a temporal problem with preferences requires to find an assignment of values to the time points that not only satisfies all the hard constraints, but also maximizes a given objective function defined over the soft constraints. Two main problem formulations taking into account preferences have been proposed in the literature: the Disjunctive Temporal Problem with Preferences (DTPP) [9], and the Valued Disjunctive Temporal Problem [10]. Both formulations, however, assume that the preferences (or costs) associated with the constraints are independent of one another. Such an assumption may prove to be too stringent in many applicative domains where dependencies among constraints could even be *conditional*: The best choice for satisfying a constraint might be independent on choices for the other constraints *given* the choices for a limited set of constraints.

Surprisingly, the problem of taking into account conditional dependencies among constraints has received little attention, so far. To the best of our knowledge, only in [8] the authors propose the Multi-Criteria extension to DTPPs (MC-DTPP).

In this paper we propose a scheduling tool dealing with preferences in a flexible way. From our point of view, the flexibility is achieved by associating tasks with durations expressed as intervals. Differently from previous approaches to flexible scheduling ([6], [7]), we do not rely on STPs, but on DTPs. More precisely, since our goal is to deal with preferences on such task durations, we propose a novel extension to DTPs which is suitable to capture a different kind of dependency.

We consider the VDTP formulation as our starting point, and complement it with a Utility Difference Network (UDN) [11] that allows for the definition of structured objective functions based on the notion of *conditional difference independence* (CDI), after which we name our extended problem formulation *CDI-VDTP*. Thanks to such conditional independencies, the computation of the utility of (partial) solutions explored during the search for an optimal solution

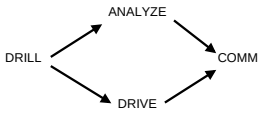

Figure 1. A segment of a rover plan.

turns out to be very similar to how probabilities are computed from a Bayesian network.

After motivating our proposal with two examples (section II), we recall background information (section III). In section IV we formally define CDI-VDTPs, and in section V we propose a way to solve them by extending a state-of-the-art algorithm for VDTPs with the optimization of UDN-based preferences. Section VI describes the main characteristics of the scheduling tool we have developed for solving this new class of problems, while section VII presents experimental results.

II. Motivating Examples

Let us present two simple examples illustrating the practical role of the extended preferences we propose.

A. Planetary Rover

First, we consider a simplified planetary rover scenario as the one discussed in [12], and assume that a mission designer is finalizing the mission that a rover has to carry out.

The mission plan has already been outlined, and Figure 1 shows a small portion of interest; edges between actions represent precedence links: Once the rover has collected a soil sample by means of the `DRILL` action, it `ANALYZE`s the sample and moves (`DRIVE`) to a position suitable for uploading (`COMM`) the collected data. The analysis and the movement could be carried on simultaneously. The designer has to decide the mode with which the activities in the plan segment have to be completed. Such a decision has to be made balancing the quality and accuracy with which some activities are performed, against the time these activities take to be successfully completed. Specifically, for each action in the plan, a set of action modes and corresponding duration intervals are defined; modes and intervals are summarized in Figure 2. Further inputs for the designer's decision making process are also global hard constraints and preferences. The designer has in fact to take into account that the action `COMM`, must be performed within a communication window, which opens over a precise period. The communication window is a hard constraint since it depends on the position of a satellite functioning as relay, and hence it is outside the control of the mission designer. Moreover, some activity modes are usually more preferred than others. For instance, it is usually preferred, and wiser, to perform a drive action in a slow mode; however, the fast mode can be used, if necessary, to avoid missing the communication window. In the tables of Figure 2, the modes of each action, considered individually, are ordered from the most preferred down to the least preferred.

DRIVE	
slow	[15, 25]
fast	[8, 13]

DRILL	
deep	[10, 13]
shallow	[5, 7]

COMM	
chn-1	[10, 15]
chn-2	[7, 10]

ANALYZE	
test-1	[7, 9]
test-2	[4, 5]
test-3	[3, 4]

Figure 2. Modes and expected durations of tasks.

S	M	T	W	T	F	S
			1	2	3	4
5	6 WKm SPm PB? MV?	7 WKt SPt MV?	8	9	10	11

Figure 3. A fragment of a working week calendar.

The challenge for the designer who has to select a mode for each action arises when we consider the different actions as being part of the same mission. In such a case, the preferred mode for an action might depend on the mode already selected for another action. For instance, a scientist would prefer to always drill with modality *deep*, because such a mode usually enables the collection of more interesting samples. On the other hand, when such samples are collected, it is preferable to analyze them with modality *test-1* which is the most accurate one. Both modes, however are very time consuming; moreover the amount of data produced by means of *test-1* mode is usually huge; this impacts the communication, since in that case the 2-channel mode *ch-2* would be preferable, even though the general preference is to use *ch-1* mode (see Figure 2).

B. Personal Calendar

Let us now consider a simple case of personal calendar management, similar to the one discussed in [8]. The idea is to help a user in the allocation of her/his tasks over a working week, taking into account both personal preferences and global constraints. In our example, the user can be involved in four different activities: work, doing some sport, going to the pub, or watching a movie. Of course, the user can be involved in only one of these activities at a time. Figure 3 shows a fragment of the work-week calendar of a sample user, whose Monday plan includes: work (*WKm*), do some sport (*SPm*), and (possibly) go to the pub (*PB*) or to the movies (*MV*). On Tuesday, the user's plan encompasses: working (*WKt*), doing sport (*SPt*), and possibly going to the movies if she/he did not go on Monday.

Similarly to the rover example, for each task there are some alternatives on how long and/or when the task should be performed. Figure 4 summarizes the possible modes for

WORK	
full	[8, 10]
part	[4, 5]

SPORT	
heavy	[2, 3]
medium	[1, 2]
light	[0, 1]

PUB	
yes	[2, 3]
no	[0, 0]

MOVIES	
opt-1	Monday 9pm
opt-2	Tuesday 9pm

Figure 4. Modes and expected durations of tasks.

each task, ordered from the most preferred to the least preferred. Note that the Pub is an optional activity as its minimal duration is 0. Moreover, the preferences associated with the Movies activity are not on the duration of the movie (which is assumed to be 2 hours), but on the day; in our case, Monday is preferred to Tuesday. Note that the preferred mode for a task might depend on the mode already selected for another one. For instance, although in general we might prefer to go to the Movies on Monday because it is cheaper, we might prefer to go on Tuesday if on Monday we work full-time and we do only a light sport activity, and hence we want to take a good rest at the Pub that evening. Working full-time on Monday could have an impact on the working activity on Tuesday, where we could prefer part-time. Besides user's preferences, the scheduling system has also to consider hard, global constraints. For instance, the user is required to work at least 30 hours a week, but at the same time the user's schedule has to include at least 5 hours of sport and at least one evening at the pub or at the movies. Thus, the user can ask the system what is the best schedule that maximizes her/his preferences while satisfying all the hard constraints.

III. Background

A. DTPs and VDTPs

A DTP is a pair $\langle \mathbf{X}, C \rangle$ where each element $X_i \in \mathbf{X}$ designates a time point, and each element $\mathcal{C}_i \in C$ is a constraint of the form $c_{i,1} \vee \ldots \vee c_{i,n_i}$, and each disjunct $c_{i,j}$ is of the form $a_{i,j} \leq X_{i,j} - X'_{i,j} \leq b_{i,j}$, with $X_{i,j}, X'_{i,j} \in \mathbf{X}$ and $a_{i,j}, b_{i,j} \in \Re$. If each constraint $\mathcal{C}_i \in C$ consists of exactly one disjunct, then the problem becomes a Simple Temporal Problem (STP), whose consistency can be checked in polynomial time with a minimum-distance algorithm for graphs such as the Johnson algorithm. [1]

A VDTP is a tuple $\langle \mathbf{X}, C, S, \varphi \rangle$ where $\mathbf{X}$, C are as in DTPs, while S and φ are defined as follows. The *valuation structure* S is a tuple $S = \langle E, \circledast, \succ \rangle$ where E is a totally ordered (w.r.t. $\succ$) set of *valuations* that can be combined with $\circledast$, a closed, associative, and commutative binary operator on E. Mapping $\varphi : C \rightarrow E$ assigns a cost $e \in E$ with (the violation of) each constraint $\mathcal{C} \in C$. In the *weighted* VDTP, structure S is $\langle \Re^+ \cup \{\infty\}, +, > \rangle$ and the function to minimize is:

$$cost(\mathbf{x}) = \sum_i \{\varphi(\mathcal{C}_i) | violates(\mathbf{x}, \mathcal{C}_i)\}$$

i.e., it is simply the sum of the costs of the constraints violated by an assignment $\mathbf{x}$ to the temporal variables $\mathbf{X}$.

We can slightly modify the definition of VDTPs in order to specify preferences instead of costs, and to define a preference value $\varphi(.)$ for each disjunct $c_{i,j}$ of each constraint $\mathcal{C}_i$. Such a modified definition is equivalent to the standard one, but we adopt it in the following because it better suits the form of our scheduling problems.

Example 1: The planetary rover example above (as the calendar example) could actually be encoded as a (modified) VDTP. The set of temporal variables $\mathbf{X}$ consists of a pair of variables for each action in the plan, denoting the start and end time of the action itself. For instance, given action `DRILL`, two variables drl_s and drl_e are included in $\mathbf{X}$. Also the communication window is encoded by means of two variables, cw_s and cw_e. In addition, a variable z is used to encode the time point used as a reference. As for the set C of constraints, we have a soft constraint for each action in the plan, for instance the `DRILL` action is associated with the following constraint:

$$\mathcal{C}_{drl} = \{[10 \leq drl_e - drl_s \leq 13] \vee [5 \leq drl_e - drl_s \leq 7]\}$$

However, the only preferences one could specify would be those informally expressed by the order of the action modes within the tables in Figure 2, that assign a preference with each satisfied disjunct independently of what disjuncts are satisfied for other constraints. Solving such a problem, thus, would lead to a solution that does not take into account the interactions among different choices in determining the preference of a global assignment.

B. Utility Difference Networks

Given a set of finite-domain variables $\mathbf{A} = \{A_1, \ldots, A_n\}$ (*attributes*), a *multiattribute utility function* $u(A_1, \ldots, A_n)$ associates a numeric value with each assignment $\mathbf{a} = a_1 \ldots a_n$ to the attributes. Utility Difference Networks (UDN) (see [11], [13]) are a graphical representation of multiattribute utility functions that exhibit strong analogies and properties with the way Bayesian Networks (BN) represent joint probability distributions.

UDNs introduce the notion of a *reference value* $\mathbf{a_i^r}$ for each attribute A_i. The notion of *reference utility function* of a subset of attributes $\mathbf{H} \subseteq \mathbf{A}$ is defined as:

$$u_r(\mathbf{H}) = u(\mathbf{H}, \overline{\mathbf{h}}^{\mathbf{r}})$$

where $\overline{\mathbf{h}}^{\mathbf{r}}$ is the reference assignment for variables $\overline{\mathbf{H}} = \mathbf{A} \backslash \mathbf{H}$. Based on u_r, the *conditional utility function* of subset

[1] In the following, sets of variables are denoted as bold-faced, capital letters, e.g., $\mathbf{X}$; variables as indexed, capital letters, e.g., $X_{i,j}$; and assignments to (sets of) variables as bold-faced lowercase letters, e.g. $\mathbf{x}$, $\mathbf{x_{i,j}}$.

$\mathbf{H1}$ given subset $\mathbf{H2}$ is defined as:

$$u_r(\mathbf{H1}|\mathbf{H2}) = u_r(\mathbf{H1}, \mathbf{H2}) - u_r(\mathbf{H2})$$

The Conditional Independence relation CDI_r and the UDNs are defined then as follows.

Definition 1: [13] Let $\mathbf{H_1}$, $\mathbf{H_2}$, $\mathbf{H_3}$ be subsets of attributes. Set $\mathbf{H_1}$ is said to be Conditionally Independent of $\mathbf{H_2}$ given $\mathbf{H_3}$ (denoted $CDI_r(\mathbf{H_1}, \mathbf{H_2}|\mathbf{H_3})$) if for any assignment $\mathbf{h_3} \in dom(\mathbf{H_3})$, $u_r(\mathbf{H_1}|\mathbf{H_2}, \mathbf{h_3}) = u_r(\mathbf{H_1}|\mathbf{h_3})$. Let $\mathbf{A}$ be a set of attributes. A Utility Difference Network (UDN) is a DAG $\mathcal{G} = (\mathbf{A}, \mathbf{E})$ such that:

$$\forall A_i \in \mathbf{A} : CDI_r(A_i, Co(A_i)|Pa(A_i))$$

where $Pa(A_i)$ are the parents of A_i, $Dn(A_i)$ are the descendants of A_i, and $Co(A_i) = \mathbf{A}\backslash(\{A_i\} \cup Pa(A_i) \cup Dn(A_i))$. UDNs decompose a multiattribute utility function into a sum in a similar way as BNs decompose a joint probability distribution into a product, namely:

$$u(\mathbf{A}) = \sum_{i=1}^{n} u_r(A_i|Pa(A_i))$$

i.e., in order to compute the utility of an assignment $\mathbf{a}$ to the attributes, it is sufficient to sum the values of the reference utility functions of each family of the UDN. The values of $u_r(A_i|Pa(A_i))$ are specified in Conditional Utility Tables (CUT).

IV. Generalizing VDTPs to CDI-VDTPs

A CDI-VDTP is an extension to VDTPs in which the evaluation structure S and mapping φ are substituted by a Utility Difference Network $\mathcal{G}$, and a utility function u over $\mathcal{G}$.

More formally, a CDI-VDTP is a tuple $\langle \mathbf{X}, C, \mathcal{G}, u \rangle$, where $\mathbf{X}$ and C are as in a standard VDTP; whereas, $\mathcal{G} = \langle \mathbf{A}, \mathbf{E} \rangle$ is a directed acyclic graph representing a Utility Difference Network such that:

- $\mathbf{A}$ is the set of network nodes (attributes). For each constraint $\mathcal{C}_i \in C$, there is an attribute $A_i \in \mathbf{A}$ s.t. $dom(A_i)$ consists of the set $\{c_{i,1}, \ldots, c_{i,n_i}\}$ of disjuncts in $\mathcal{C}_i$;
- $\mathbf{E}$ is a set of oriented edges $\langle A_i, A_j \rangle$ such that $A_i, A_j \in \mathbf{A}$. The edges in $\mathbf{E}$ describe the dependencies among the attributes over which one is interested in finding an assignment that maximizes the utility u. For instance, the edge $\langle A_i, A_j \rangle$, means that the selection of a value for A_i (disjunct for constraint $\mathcal{C}_i$) (possibly) affects the utility of the value selection for A_j (i.e., disjunct for $\mathcal{C}_j$) for maximizing the global utility.

Thanks to the properties of UDNs, the utility function u is compactly represented as a set of reference utility functions $u_r(A_i|Pa(A_i))$ for each $A_i \in \mathbf{A}$. In the following, we shall need to compute the maximum utility achievable given an assignment $\mathbf{h}$ to a subset $\mathbf{H} \subseteq \mathbf{A}$ of variables. In analogy with the Most Probable Explanation (MPE) for Bayesian Networks, we define the Most Preferred Completion (MPC) of an assignment $\mathbf{h}$ as:

$$MPC(\mathbf{h}) = \arg\max_{\overline{\mathbf{h}}}(u(\mathbf{h}, \overline{\mathbf{h}})).$$

Namely, $MPC(\mathbf{h})$ is the assignment $\overline{\mathbf{h}}$ that completes $\mathbf{h}$ and yields a maximal utility. The computation of the MPC of a hypothesis $\mathbf{h}$ can be performed by adapting algorithms for computing the MPE of some evidence in a BN. We have chosen to use the well known Jointree algorithm, which is particularly well-suited to the reuse of cached partial results for the incremental computation of the MPC of a new hypothesis $\mathbf{h}'$ (see the experiments in section VII).

Example 2: Let us consider again the planetary rover scenario. To model the preference values associated with its constraints, we have to consider the dependencies among them. In particular, we can assume that `DRILL` does not depend on any previous action, but it does influence `ANALYZE`, which in turn influences `COMM`. On the other hand, `DRIVE` can be considered as independent of the other actions. Relying on these assumptions, in Figure 5 we sketch the UDN for this problem: Each node corresponds to a constraint in C (including the hard constraint on the communication window); edges between nodes denote preference dependencies; in addition, in analogy to a Bayesian network, each node is associated with a CUT that defines the preferences for a constraint given its parent nodes.

In this particular case, the utility network has three components. Two components are $\mathcal{C}_{drv}$ and $\mathcal{C}_{cw}$, representing the constraints associated with the drive action and the communication window, respectively. Being roots, a utility value is directly assigned to each of their disjuncts. For instance, the utility table associated with $\mathcal{C}_{drv}$ states that *slow* is generally preferred to *fast*. In addition, since the constraint about the communication window is hard, it is associated with two "fake modes", *satisfied* and *unsatisfied*, and the latter one has utility $-\infty$, meaning that any solution that violates the communication window constraint is not acceptable. The root of the third component is $\mathcal{C}_{drl}$, which influences the constraint $\mathcal{C}_{anl}$ associated with the analysis action. In this case, the utility associated with each disjunct in $\mathcal{C}_{anl}$ depends on the disjuncts that have been selected for its parents (only $\mathcal{C}_{drl}$ in this example). This results in a CUT which looks like a Conditional Probability Table in a Bayesian network. The particular table in the figure is to be interpreted as follows; independently of how deep the drill operation is, there is a strong preference in performing *test-1*; however, if the *test-1* is not possible, *test-2* should be preferred when the drill action was *deep*, whereas *test-3* should be preferred when the drill was *shallow*. Similarly, $\mathcal{C}_{anl}$ affects $\mathcal{C}_{com}$ (i.e., the constraint associated with the communication). Note, in this case, that when the analysis was carried out with mode *test-1*, the usage of mode *ch-1* is

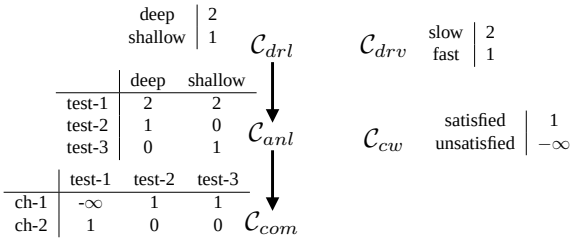

Figure 5. The Utility Difference Network for the rover example.

solve-CDI-VDTP$(\mathbf{h}, \mathbf{mpc}, \overline{\mathbf{H}}, lwb, \Delta)$
1. $util \leftarrow u(\mathbf{h}, \mathbf{mpc})$
2. **if** $util < lwb$ **then**
3. **return**
4. **end if**
5. **if** $\overline{\mathbf{H}} = \emptyset$ **then**
6. **if** $util > lwb$ **then**
7. $\Delta \leftarrow \emptyset$
8. $lwb \leftarrow util$
9. **end if**
10. $\Delta \leftarrow \Delta \cup \{\mathbf{h}\}$
11. **return**
12. **end if**
13. $A_i \leftarrow$ **select-attribute**$(\overline{\mathbf{H}})$; $\overline{\mathbf{H}}' \leftarrow \overline{\mathbf{H}} - \{A_i\}$
14. $modes \leftarrow dom(A_i)$
15. **while** $modes \neq \emptyset$ **do**
16. $m \leftarrow$ **select-mode**$(modes)$; $modes \leftarrow modes \backslash \{m\}$
17. $\mathbf{h}' \leftarrow \mathbf{h} \cup \{A_i \leftarrow m\}$
18. **if consistent**$(\mathbf{h}')$ **then**
19. **solve-CDI-VDTP**$(\mathbf{h}', MPC(\mathbf{h}'), \overline{\mathbf{H}}', lwb, \Delta)$
20. **end if**
21. **end while**

Figure 6. The solve-CDI-VDTP algorithm.

practically forbidden. On the other hand, the usage of *ch-1* should be preferred when the analysis was conducted either with *test-2* or *test-3* mode.

It is worth noting that at this stage of development, we assume that the utility values indicated in these tables result from information provided by the problem designer, who takes into account features of the rover that are not explicitly addressed by the temporal problem. For example, the preference on a slow drive could be motivated by security reasons; whereas the preference of the usage of *ch-1* to *ch-2* could depend on the fact that the second mode is more resource consuming.

V. Solving CDI-VDTPs

To solve a CDI-VDTP problem we adopt a strategy similar to the one proposed in [10]. The strategy recursively proceeds in a depth-first manner, and branches are pruned whenever their utility is guaranteed to fall below the cost of the best (i.e., maximal) solution found so far.

Our search strategy is outlined in the algorithm in Figure 6. The algorithm takes as inputs:

- $\mathbf{h}$: a (partial) assignment of modes to a subset of attributes $\mathbf{H}$, i.e., a (partial) hypothesis;
- $\mathbf{mpc}$: the Most Preferred Completion of $\mathbf{h}$;
- $\overline{\mathbf{H}} = \mathbf{A} \backslash \mathbf{H}$ is the set of attributes whose mode has not been assigned yet;
- lwb: the utility of the best solution found so far;
- Δ: the set of all the best solutions found so far.

It is worth noticing that, while the first three arguments are passed by value, the last two arguments are passed by reference. Thereby, any change made during an invocation of **solve-CDI-VDTP** impacts all instances of the algorithm possibly active on the stack. In particular, when the search terminates Δ contains the set of best solutions and lwb their utility.

At each invocation, the algorithm determines the upper bound of the utility achievable by completing the current (partial) solution $\mathbf{h}$ (line 1), and checks whether it is lower than the best one so far (line 2); if yes, such a branch is not useful so it is pruned with the return statement. Otherwise, the algorithm checks whether there are still variables to be assigned (line 5): if $\overline{\mathbf{H}}$ is empty, then all attributes have been assigned and $\mathbf{h}$ is a complete solution. At this stage, the algorithm checks whether the new complete solution is better than any other solution found so far (lines 6-9); in the positive case, lwb is updated to be the utility of $\mathbf{h}$, and Δ is emptied as all the solutions found so far were not optimal. In any case, $\mathbf{h}$ is added to Δ (line 10).

In case $\mathbf{h}$ is still a partial solution, the algorithm tries to get closer to a solution by selecting an attribute A_i from $\overline{\mathbf{H}}$ (line 13). Then the algorithm considers each mode m in $dom(A_i)$ (lines 15-21), in the order determined by function *select-mode* (line 16), and generates new hypotheses from them. In particular, for each $m \in dom(A_i)$, a new hypothesis $\mathbf{h}'$ is obtained by adding the assignment $A_i \leftarrow m$ to $\mathbf{h}$. The temporal consistency of the new hypothesis $\mathbf{h}'$ is then verified by means of function *consistent* (line 18), that performs a consistency check on the STP induced by the modes in $\mathbf{h}'$. Finally, function *solve-CDI-VSDP* is recursively invoked over the new hypothesis $\mathbf{h}'$ and the new set of unassigned variables $\overline{\mathbf{H}}'$ (line 19).

The choice of the next attribute/mode to assign (calls to *select-attribute* and *select-mode*) can benefit from the heuristics established for DTP solving [14], such as conflict-directed backjumping, removal of subsumed variables, semantic branching, and no-good recording. However, in addition to such standard techniques, the choice of the next mode m to try for an attribute A_i can be determined by exploiting $\mathbf{mpc}$. In particular, if $\mathbf{mpc} = MPC(\mathbf{h})$ assigns mode m_{mpc} to attribute A_i which is chosen next, that should be the first mode to try for A_i, since it maximizes the utility according to the UDN. Note that, in general,

given a hypothesis $\mathbf{h}$ there may be several completions that maximize the utility, that may assign different modes to A_i. If the MPC computation is able to return all of them, the calls to *select-mode* should return them before the other modes of A_i.

Since there is no room for a detailed computational analysis of the algorithm, we refer to [15] where we analyze a similar branch-and-bound approach, and show that the complexity is exponential in the number of the attributes.

Example 3: Let us go back to the simple mission plan given in Figure 1, and consider a first scenario in which the communication window opens at 35 and closes at 50 time units. Such a global constraint is not particularly stringent for the plan. In fact, the CDI-VDTP algorithm finds a schedule satisfying all the preferences with their maximal utility; more precisely, the modes associated with actions are as follows: ⟨ `DRILL`: *deep*⟩, ⟨ `DRIVE`: *slow*⟩, ⟨ `ANALYZE`: *test-1*⟩, and ⟨ `COMM`: *ch2*⟩. It is easy to see that even when all these actions take the longest possible time, the communication action would occur from time 38 to 48. On the other hand, if we consider the communication window [30, 45], the algorithm has to save some time, e.g., by performing a shallow drill. The lack of time has also an impact on the usage of channel *ch2* that, despite not being the preferred one in case of a shallow drill, is required to meet the window. Thus, in this scenario the best assignments would be: ⟨ `DRILL`: *shallow*⟩, ⟨ `DRIVE`: *slow*⟩, ⟨ `ANALYZE`: *test-3*⟩, and ⟨ `COMM`: *ch2*⟩.

VI. Scheduling Tool

We have implemented the approach described in this paper as a scheduling tool written in Perl 5.16. The main inputs to the tool are:

- a *UDN* that defines the *action types* (e.g., *analyze-type*), their *modes* (e.g., the *test-1* mode for analyze, which takes $[7, 9]$ time units), and the (conditionally independent) preferences associated with such modes
- a *plan* that defines the actual actions (e.g., there may be two actions *ANALYZE1* and *ANALYZE2* of type *analyze-type*) and the precedence constraints among them (e.g., action *ANALYZE1* must follow *DRILL1*)
- a specific *case*, specifying additional constraints that only apply to the scheduling problem under consideration; such constraints consist in the time of occurrence of certain events (e.g., the *ANALYZE1* action must end within 20 time units from the start of the mission) or modes associated with actions (e.g., the *ANALYZE2* action must be performed in *test-2* mode)

Both the *plan* and *UDN* are represented in memory as directed graphs by exploiting the *Graph.pm* module from CPAN [2], a very flexible and feature rich Perl implementation of the graph data structure. On disk, *plan* and *UDN* are conveniently encoded in the YAML format (Yet Another Markup Language), which achieves a fairly good level of human-readability without being too verbose. A fragment of the YAML representation of the UDN for the Rover domain in shown in Figure 7; note that the CUT table specifies the conditional preferences of the mode of *ANALYZE* given the mode of *DRILL* (as given in Figure 5). A companion tool of the scheduler, written in Java, is able to automatically generate specific *case*s from the *UDN* and *plan* files, thus allowing the creation of test cases of different sizes for the experiments (section VII).

[2]Comprehensive Perl Archive Network, a global archive of user-contributed, open source Perl modules.

```
analyze-type:
      agent-type: rover
      modes:
      - {label: test-1, lb: 7, ub: 9, rank: 3}
      - {label: test-2, lb: 4, ub: 5, rank: 2}
      - {label: test-3, lb: 3, ub: 4, rank: 1}
      conditioned-by:
      - drill-type:
      cut:
      - [2, 2]
      - [1, 0]
      - [0, 1]
```

Figure 7. Fragment of YAML encoding the UDN for the Rover Domain.

While *Graph.pm* is used to represent and manipulate plans and UDNs, the STPs that are built during the search process and checked for consistency (line 18 in Figure 6) are represented using another module from CPAN, namely the *Boost::Graph.pm* module. Such a module is a Perl wrapper for the well known C++ Boost Graph Library (BGL), which makes the consistency checks much faster than in pure Perl.

As an output, the tool computes *all* of the preferred assignments of modes to actions which satisfy the constraints and preferences specified in the inputs *UDN*, *plan*, and *case*. Each of such assignments is actually a flexible schedule, where the start of each action corresponds to an interval of allowed values. The derivation of a specific schedule from the output of the tool can be made efficiently off-line (e.g., by picking as the start time of each action the lower bound of its interval), or dynamically during plan execution (e.g., deciding the start time of an action only after the actions preceding it have terminated).

VII. Experimental Results

Since the CDI-VDTP is a new type of problem, not directly comparable with previous approaches on VDTPs and DTPPs, we set out experiments for assessing the feasibility and scalability of our methodology. The tests have been run on a virtual machine running Linux Ubuntu 12.04, equipped with an i7 M640 CPU at 2.80 GHz, and 4 GB RAM.

We have considered the two domains sketched in the examples, namely Planetary Rover (*RV*) and Calendar (*CAL*). Figure 8 shows the UDN structures we have adopted to generate test cases of increasing sizes for the two domains.

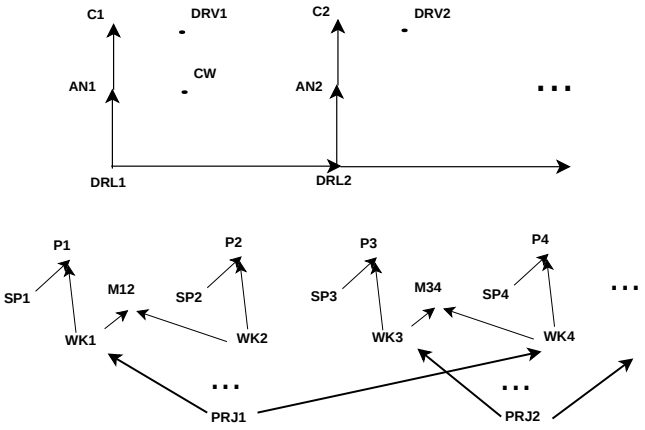

Figure 8. The UDNs for the Rover Domain (top) and Calendar Domain (bottom).

In particular, for the RV domain, the UDN consists of a number of repetitions of the structure discussed in the example above (Figure 5), connected by influences between the drill actions; namely, we have made the assumption that the choice of the mode for a drill (deep, shallow) will influence the choice for the next drill (e.g., because drills are made close to each other, and similar choices are usually made). For the CAL domain, we have considered a structure capturing some of the possible influences mentioned in the example: the time spent working and doing sport influences the willingness to go to the pub, and the choice on when to go to the movies. Then, we have connected such structures to projects (*PRJ*) that we can decide to work on, that influence the daily workload (and can overlap).

We have generated three test sets *TS1*, *TS2* and *TS3* of increasing scale for each domain. Table I reports the following characteristics of each test set:

- length *len* of the problems, expressed in terms of number of drills (RV domain) or days (CAL domain)
- number *#tsks* of tasks to allocate and number *#vars* of variables
- number *#c-cfg* of temporal constraints for any specific configuration of modes and *#c-cs* of additional constraints that change for each test case
- *#edges* of the UDN describing the dependencies among constraints

The number of tasks *#tsks* corresponds to the number of disjunctive temporal constraints in the test cases; for example, a case belonging to test-set *RV1.TS1* (i.e., test set *TS1* of domain *RV*) needs to assign a mode to 40 tasks, namely the four tasks (*DRILL*, *ANALYZE*, *DRIVE*, and *COMM*) associated with each of the 10 drills. Similarly, a case belonging to test-set *CAL.TS1* needs to assign a mode to 37 tasks: three tasks for each day (*SPORT*, *WORK*, and *PUB*), plus five movies and two projects. Note that, considering the pairs of corresponding test sets *RV.TSi* ($i = 1, \ldots, 3$) in the two domains, the number of tasks (and thus of variables, since each task requires a start and an end variable) is higher for *RV*.

Table I
CHARACTERISTICS OF THE TEST SETS *TSi* FOR THE ROVER (TOP) AND CALENDAR (BOTTOM) DOMAINS.

RV	#len	#tks	#vars	#c-cfg	#c-cs	#edgs
TS1	10	40	84	142	7	29
TS2	20	80	164	282	14	59
TS3	30	120	244	422	21	89

CAL	#len	#tsks	#vars	#c-cfg	#c-cs	#edgs
TS1	10	37	76	119	7	42
TS2	20	72	146	236	14	94
TS3	30	107	216	351	21	141

Table I also reports the number *#c-cfg* of constraints that apply to any configuration of task modes in the given test set. For the *RV* domain, they include a min and max duration for each task, plus precedence constraints and two constraints defining the communication window (e.g., 142 constraints for the test set *RV.TS1*). Finally, Table I reports the number of edges in the UDNs used in the test sets. This statistic is significantly higher for the *CAL* domain, witnessing a higher complexity of the preference structure w.r.t. the *RV* domain.

We have run the test cases in each test set. Table II shows the average of the following statistics:

- *time/sol*: time to compute a solution;
- *#sols*: number of preferred solutions found.

Note that each test set has been solved both with and without activating caching in the algorithm used in the computation of the Most Preferred Completions (see section IV). Moreover, for each test case we have computed all of the preferred solutions.

Looking at the results obtained *with* the use of caching, we observe a good performance in both domains. Even the test sets *RV.TS3* and *CAL.TS3* that have hundreds of variables, constraints and influence edges between preferences can find each preferred solution in a very reasonable amount of time. In general, even if the corresponding test sets *TSi* have more tasks, variables and constraints in the *RV* domain, the performance achieved for *RV* is generally better than that achieved for *CAL*. This is easily explained by the fact that *CAL* has more complex preference structures, that make the many MPC computations more expensive.

Finally, we note that caching reduces very significantly the time needed for finding a solution in both domains. In particular, in the *CAL* domain, such a reduction is between 67% and 72% for the three test sets; in the *RV* domain it is between 57% and and 62%.

Table II
AVG TIME/SOL (SEC), AND NUMBER OF SOLS FOR THE ROVER (TOP) AND CALENDAR (BOTTOM) DOMAINS.

RV	**TS1**		**TS2**		**TS3**	
cache	**yes**	**no**	**yes**	**no**	**yes**	**no**
time/sol	2.6	6.1	8.8	23.3	15.4	40.9
#sols	3		3.8		3.7	

CAL	**TS1**		**TS2**		**TS3**	
cache	**yes**	**no**	**yes**	**no**	**yes**	**no**
time/sol	2.1	6.4	11.8	40.0	22.2	77.9
#sols	4.2		2.1		2.9	

VIII. CONCLUSIONS

In this paper we addressed the problem of synthesizing a flexible schedule taking into account soft constraints (i.e., preferences). Our approach goes beyond previous methodologies on disjunctive temporal problems since it is capable of dealing with temporal preferences that are only *conditionally* independent of one another.

The paper contributions are twofold. On the one side, the paper presented a formal methodology, named CDI-VDTP, which extends the VDTP formulation [10] of temporal problems with the notion of Conditional Difference Independence. CDI-VDTP enables a user to take advantage of the causal dependencies between the preferences associated with the constraints, and to define an objective function shaped over a Utility Difference Network (UDN), in which each node corresponds to a constraint and (oriented) edges between nodes represent causal dependencies. Solving a CDI-VDTP, thus, consists in computing solutions whose utility is optimal; this can be achieved by exploiting algorithms which are similar to those used for computing probabilities in a Bayesian network, but applied to the UDN.

On the other side, the paper also presented a tool which, relying on a branch-and-bound algorithm, enables a user to submit and solve CDI-VDTPs by exploring the space of possible solutions. Results collected by a preliminary implementation have been discussed, and show that the proposed solution is actually feasible even for quite large problems.

As a future work, we intend to further extend the CDI-VDTP formulation with the addition of a set of variables that, although included within the UDN, are not associated with temporal constraints. The rationale would be to explicitly model via these variables aspects of the domain under consideration that might affect the preference values of a subset of constraints. For instance, in the planetary rover scenario, the level of battery power could be represented explicitly within the UDN by means of a specific variable; such a variable could then affect the duration of actions such as *drive*, depending on the assumed level of power. Tasks such as planning and diagnosis could exploit such a richer formulation to create expectations or verify hypotheses.

Acknowledgements

This work was partially supported by the *Accountable Trustworthy Organizations and Systems (AThOS)* project, funded by Università degli Studi di Torino and Compagnia di San Paolo (CSP 2014).

REFERENCES

[1] R. Dechter, I. Meiri, and J. Pearl, "Temporal constraint networks," *Artificial Intelligence*, vol. 49, pp. 61–95, 1991.

[2] K. Stergiou and M. Koubarakis, "Backtracking algorithms for disjunctions of temporal constraints," *Artificial Intelligence*, vol. 120, no. 1, pp. 81–117, 2000.

[3] A. Oddi and A. Cesta, "Incremental forward checking for the disjunctive temporal problem," in *Proc. ECAI*, 2000, pp. 108–112.

[4] N. Roos and C. Witteveen, "Diagnosis of simple temporal networks," in *Proc. ECAI*, 2008, pp. 593–597.

[5] R. Micalizio and G. Torta, "Diagnosing delays in multi-agent plans execution," in *Proc. ECAI)*, 2012, pp. 594–599.

[6] N. Policella, S. F. Smith, A. Cesta, and A. Oddi, "Generating robust schedules through temporal flexibility." in *ICAPS*, vol. 4, 2004, pp. 209–218.

[7] N. Policella, A. Cesta, A. Oddi, and S. F. Smith, "From precedence constraint posting to partial order schedules," *AI Communications*, vol. 20, no. 3, pp. 163–180, 2007.

[8] M. D. Moffitt, B. Peintner, and N. Yorke-Smith, "Multi-criteria optimization of temporal preferences," in *Proc. of CP06 Workshop on Preferences and Soft Constraints (Soft06)*. Citeseer, 2006, pp. 79–93.

[9] L. Khatib, P. H. Morris, R. A. Morris, and F. Rossi, "Temporal constraint reasoning with preferences," in *IJCAI*, 2001, pp. 322–327.

[10] M. D. Moffitt, "On the modelling and optimization of preferences in constraint-based temporal reasoning," *Artificial Intelligence*, vol. 175, no. 7-8, pp. 1390–1409, 2011.

[11] R. I. Brafman and Y. Engel, "Directional decomposition of multiattribute utility functions," in *Algorithmic Decision Theory*. Springer, 2009, pp. 192–202.

[12] J. L. Bresina, A. K. Jónsson, P. H. Morris, and K. Rajan, "Activity planning for the mars exploration rovers." in *In 15th International Conference on Automated Planning and Scheduling (ICAPS 2005*, 2005, pp. 40–49.

[13] R. I. Brafman and Y. Engel, "Decomposed utility functions and graphical models for reasoning about preferences." in *AAAI*, 2010.

[14] I. Tsamardinos and M. E. Pollack, "Efficient solution techniques for disjunctive temporal reasoning problems," *Artificial Intelligence*, vol. 151, no. 1, pp. 43–89, 2003.

[15] R. Micalizio and G. Torta, "Explaining interdependent action delays in multiagent plans execution," *Journal of Autonomous Agents and Multi-Agent Systems*, 2015.